Balanced distributed search trees do not exist

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Theoretische Informatik

Brigitte Kröll
Peter Widmayer

Balanced Distributed Search Trees Do Not Exist

May 1995

ETH Zürich
Departement Informatik
Institut für Theoretische Informatik
Prof. Dr. P. Widmayer

Brigitte Kröll
Computer Science Department
Federal Institute of Technology
CH-8092 Zürich, Switzerland
e-mail: kroell@inf.ethz.ch

Peter Widmayer
Computer Science Department
Federal Institute of Technology
CH-8092 Zürich, Switzerland
e-mail: widmayer@inf.ethz.ch

This report is also available via anonymous ftp from ftp.inf.ethz.ch
as doc/tech-reports/1995/233.ps.

Abstract

This paper is a first step towards an understanding of the inherent limitations of distributed data structures. We propose a model of distributed search trees that is based on few natural assumptions. We prove that any class of trees within our model satisfies a lower bound of $\Omega(\sqrt{m})$ on the worst case height of distributed search trees for m keys. That is, unlike in the single site case, balance in the sense that the tree height satisfies a logarithmic upper bound cannot be achieved. This is true although each node is allowed to have arbitrary degree (note that in this case, the height of a single site search tree is trivially bounded by one). By proposing a method that generates trees of height $O(\sqrt{m})$, we show the bound to be tight.

1 Introduction

Distributed data structures have attracted considerable attention in the past few years. From a practical viewpoint, this is due to the increasing availability of networks of workstations. These networks offer an enormous computing power not only for distributed algorithms, but also for efficient storage and retrieval. Since collections of data become larger and larger, and efficient access is still a bottleneck in quite a few applications, it is useful to know how to efficiently maintain data in a distributed environment. From a theoretical perspective, the appeal of this question comes from the fact that the distributed setting turns out to be substantially different from its classical (single site) counterpart, and that it poses challenging new problems.

The seminal work on distributed linear hashing (LH*, Litwin et al. [5]) has been followed by suggestions based on distributed extendible hashing (Devine [1], Vingralek et al. [7]), distributed random binary search trees (DRT, Kröll et al. [2]), and a distributed variant of B-trees (RP*, Litwin et al. [6]). In some essential aspects of the model of distributed data structures (such as scalability requirements) and of the measure of efficiency, all of this work agrees with the proposal by Litwin et al. [5]. Nevertheless, this whole research area is still far from an accepted setting that models all essential features of the distributed world well enough. In particular, it is by no means clear what performance could possibly be achieved in a particular setting.

In this paper, we are interested primarily in lower bounds on the performance of distributed data structures. We therefore ignore distributed hashing methods (Devine [1], Litwin et al. [5]), because they fail to guarantee good worst-case efficiency, and we restrict ourselves to *general* distributed search structures that are only based on key comparisons. Within this framework, we naturally limit ourselves to search trees. Up until today, two distributed search tree structures have been proposed in the literature, a distributed random tree DRT [2] and a distributed variant of a B-tree [6]. None of both is satisfactory from a theoretical point of view: The DRT is not able to narrowly bound the length of a longest search path, and the distributed B-tree variant cannot confine a search on a path from the root to a leaf — it may err and need to go back up in the tree, and therefore the logarithmic bound on its height does not imply a logarithmic bound on the search time. In order to better understand the inherent limitations of distributed search trees, we study classes of distributed leaf search trees of a certain type that can be viewed as a natural generalization of the usual single site leaf search trees. We will show that no class of trees of this type can guarantee a logarithmic bound on the path length. This puts the distributed case in contrast with the traditional single site case, where we know how to balance search trees.

More precisely, we prove that any class of trees within our model satisfies a lower bound of $\Omega(\sqrt{m})$ on the worst-case height of distributed search trees for m keys. We give a matching upper bound by sketching a method that generates trees of exactly that height.

This paper is organized as follows. Section 2 discusses the problem of scalable, distributed access structures and reports on crucial aspects of the proposals in the literature. Section 3 defines

the model of distributed data structures on which we base our lower bound proof. In Section 4, we prove the lower bound; in Section 5 we propose a distributed data structure that leads to a matching upper bound. Section 6 discusses implications of our result.

2 The distributed search problem

We want to study the inherent efficiency limitations of dynamic, distributed data structures. Therefore, we ignore most of the problems that a distributed data structure might encounter in practice (such as faulty communication, for instance) and resort to a simple model. We will show that our lower bound holds already in this simple model. Like all schemes proposed recently in the literature, we assume that a given set of sites (processors) is connected by a point-to-point network. Each site in the network is either a *server* that stores data or a *client* that initiates requests. Sites communicate by sending and receiving messages. A site can send a message to any site, given the identifier of the destination site (we assume site identifiers to be unique network addresses), in a single communication step. The network communication is free of errors. Each site buffers all incoming messages and guarantees that each message is handled in finite time after its arrival. All clients together operate on a (presumably huge) common file that must be distributed over the servers dynamically. Each server can store a single block of at most b data items, for a fixed, constant number b (we will therefore not include b in the asymptotic efficiency bounds to be derived later in the paper). The maintenance of the data within each server's storage space is irrelevant; in particular, it does not matter whether the data local to a server are stored in main memory or on external storage. The distributed data structure determines the distribution of the data over the servers; there are no preconditions as to where the data can be stored. The data is viewed as a set of keys that are drawn from a linearly ordered universe, where a key represents an arbitrary object whose nature is irrelevant for access purposes. In this set, a client triggers an insert or search operation for a key by sending a message to a server. In the best case, this server can perform the operation on its local data. In general, however, it may be necessary for the server to forward the message to some other server (and so on), because the distribution of the keys among the servers changes dynamically according to the set of inserted keys. We disregard deletions, in line with the literature so far.

For the purpose of measuring efficiency, we view the given network to be a complete graph with bidirectional links. We measure the efficiency of operations solely in the number of messages exchanged between sites. That is, the length of a message and the topology of a connected network that may in reality underly the complete communication graph do not influence the cost of a message transmission (this measure of efficiency was already proposed by Litwin et al. [5]).

The distributed random tree

To illustrate some essential aspects of distributed search trees, let us now briefly discuss the distributed random tree DRT; for a full description of this method see [2].

DRT distributes data according to a virtual, global, complete binary random leaf search tree T. Each node of T is uniquely assigned to one server. The clients and servers have some possibly obsolete knowledge of some part of T. In any case, the knowledge of a server is at least sufficient for guiding a search: For an interior node assigned to a server, the server knows about the node and its children, in the sense that it can correctly forward a key in a search operation to its left or its right child (see Fig. 1). For a leaf assigned to a server, the server stores all inserted keys in the data set whose search paths end at that leaf. Only one leaf is assigned to a server at any given time. That is, a server stores the block of data items whose keys lie in the key interval of the server's leaf, plus a subtree of T to help guide a search. In order to guarantee that no server is forced to store

more than b keys, the virtual, global tree T grows at the leaves, exactly like a random search tree, and the assignment of nodes to servers is adjusted as follows: Whenever a leaf is transformed into an interior node with two leaves as children, the new interior node remains assigned to its server, one of the new leaves is assigned to the same server, and the other leaf is assigned to a new server. Now, conceptually, a search for key k traverses the search path defined by k in T and ends in a leaf assigned to the server that holds the data with which the request can be answered. The traversal of the path in the virtual, global tree T is realized by forwarding the request from one server to the next on this path. While in the DRT method the knowledge that active clients and servers

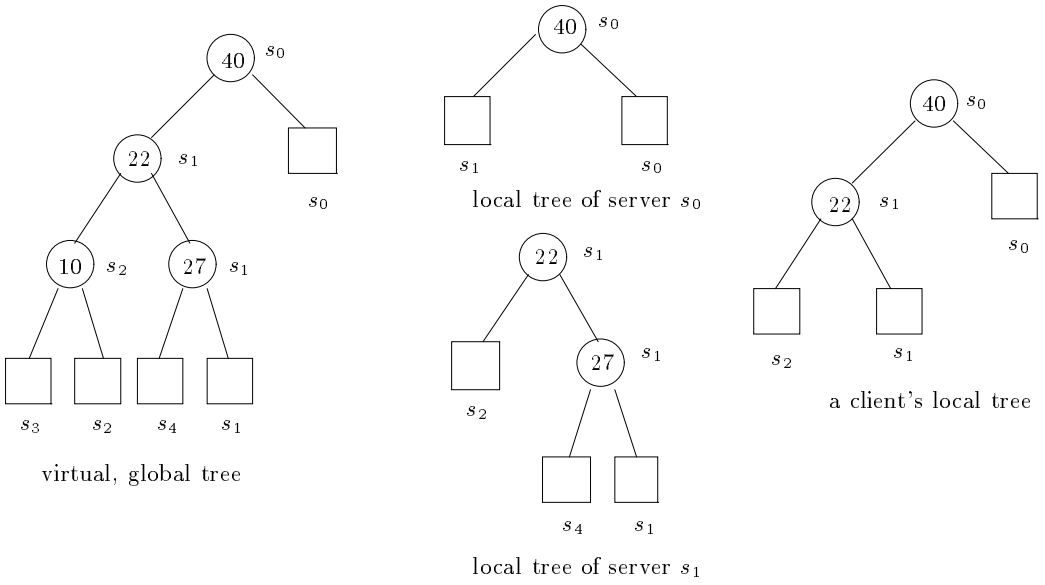

Figure 1: An illustration of the DRT method (it shows for each node the assigned server).

often have about the virtual, global tree implies that a leaf will often be reached with none or only a few forward steps, in the worst case a forward step may actually occur in each position on the search path in which the server changes. In addition, the virtual, global tree may degenerate to a single path on which different interior nodes are assigned to different servers. As a consequence, in the worst case the number of forward steps of a request is linear in the number of servers that currently participate in the method.

Distributed search trees in general

Our interest in this paper is whether we can avoid this worst case; ideally, we would like to define a class of distributed search trees with logarithmic height. To state the problem precisely, we need a suitable definition of the term *search tree* for the distributed case. Let us recall the single site case. The search tree reflects the partition of the universe into intervals according to the set of keys: Each node represents a key interval, with the root representing the universe, and the children of a node representing the intervals that partition the node's interval. A search operation maintains the invariant that a key visits a node only if it lies in the node's interval; it does so by starting the search at the root and progressing at each node to the appropriate child. Let us call this part of the search tree behavior the *straight guiding property.*

In the distributed case, however, we count the number of messages instead of counting the steps in the tree traversal as usual. That is, we want to guarantee the straight guiding property for these messages. We restrict our attention to the case in which only one server (and not more than one) is associated with a node; this is the case in all methods suggested so far in the literature. We call such a tree a *distributed search tree.* Our goal is to ensure that a key that arrives at a server belongs to the set of keys that the server represents (that is the union of the key intervals of all nodes with which the server is associated). This is not trivially satisfied in a distributed environment: The virtual, global tree changes over time, including the association of servers to nodes, and not all sites know the respective current tree. Therefore, a key may be sent to a server based on potentially obsolete information on the sender's side, and thus we need to make sure that at any time, the receiving server can straightly guide all keys that it could correctly receive (and hence guide) at some previous time. This distributed version of the straight guiding property is the key requirement that we impose on distributed search trees; it limits our freedom in modifying the search tree structure itself, and in associating servers with nodes of the search tree. Let a *stable distribution method* denote a method that guarantees that the distributed straight guiding property is satisfied.

Observe that the virtual, global tree of the DRT method possesses the distributed straight guiding property. Hence, the random search tree method naturally generalizes to a stable distribution method, namely the DRT method. Does the same hold for some class of balanced search trees, in the sense that each path has only logarithmic height? If it does not, can we still define a class of balanced, distributed search trees?

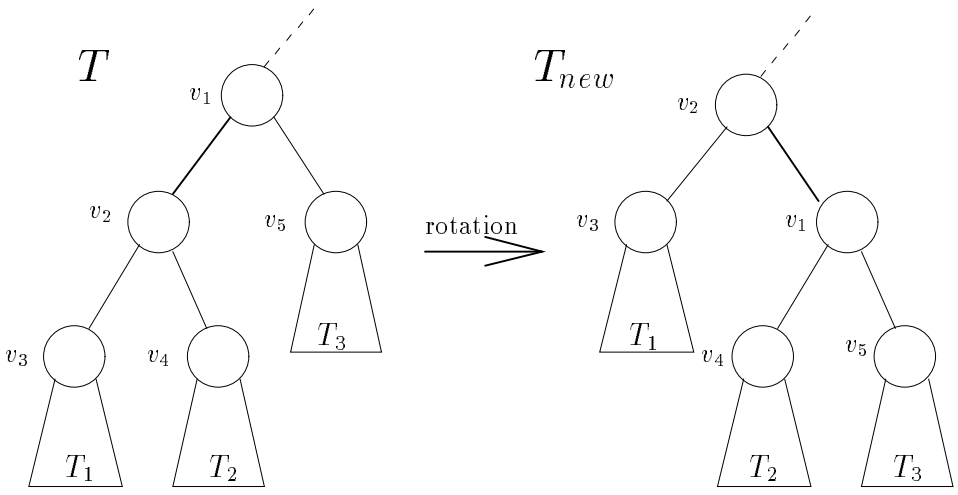

Figure 2: The effect of a distributed rotation.

To get some insight into where the difficulties might lie, let us for a moment try to distribute a file according to a search tree that is rebalanced by means of rotations (see Fig. 2). Assume that server s_1 is assigned to v_1 and server $s_2 \neq s_1$ to v_2 in T, which is transformed to T_{new} by the rotation shown in Fig. 2. Now, let us try to reassign servers to nodes in T_{new} in a way that guarantees the straight guiding property. Note that the set of keys visiting v_1 in the search tree T before the rotation is a superset of the set of keys visiting v_1 in the search tree T_{new} after the rotation. Thus, server s_1 may receive a request with a key whose search path ends in T_1, since s_1 is assigned to v_1 in T. Now, by simply reassigning the servers s_1 and s_2 to the nodes v_1 and v_2 in T_{new}, we cannot guarantee the straight guiding property. In this case, we must assign one of the servers to a node on the path from the root to v_2 (except v_2 itself) in T_{new}. Because this affects nodes outside the scope of the rotation, this example indicates that it is impossible to directly make use of rotations for balancing a distributed search tree while guaranteeing the straight guiding property.

In the following, we will show that distributed search trees that are balanced (in the sense that the tree height satisfies a logarithmic upper bound) according to a stable distribution method do not exist. This is true although we allow each node in the tree to have arbitrary degree (note that in this case, the height of a single site search tree is trivially bounded from above by 1), and although each insert operation may change the virtual, global tree entirely (provided that the straight guiding property remains satisfied). Our results show that no stable distribution method can generate distributed search trees with a height of $o(\sqrt{m})$, where m is the number of stored keys; a bound of $O(\sqrt{m})$, however, can be achieved.

3 The model

Let us now formally describe the model that underlies our lower bound on the performance of stable distribution methods. It largely makes the preceding discussion more precise.

Let $\mathcal{K}$ be a totally ordered universe of keys with the following two properties. First, between any two different keys, there is an infinite number of other keys, and second, no minimum and no maximum key exist. Examples for $\mathcal{K}$ are the set of rational and the set of real numbers. We define $-\infty$ (resp. ∞) to be smaller (resp. bigger) than all keys in $\mathcal{K}$.

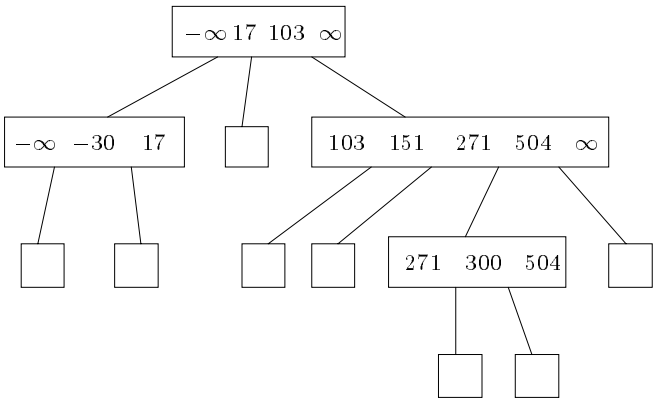

Figure 3: An example search tree.

We limit our discussion to the class of leaf search trees for $\mathcal{K}$ with the following notation (for an example, see Fig. 3): Each node represents a key interval $[k_0, k_r)$ of all keys bigger than or equal to k_0 and smaller than k_r ($k_0, k_r \in \mathcal{K} \cup \{-\infty, \infty\}$) and stores the interval boundaries k_0 and k_r. For the root node, we have $k_0 = -\infty$ and $k_r = \infty$. As usual, each interior node stores the

partition of its key interval $[k_0, k_r)$ represented by its children; it does so by storing $r-1$ routers $k_1, \ldots, k_{r-1} \in \mathcal{K}$, where $r \geq 2$ and $k_0 < k_1 < \ldots < k_r$. A search in the tree starts at the root and is guided by the routers on a unique search path down the tree to a leaf: For key $k \in \mathcal{K}$, the search follows the ith pointer if $k_{i-1} \leq k < k_i$ holds. Each leaf of the tree represents the data block of all data within its key interval; that is, the tree is merely an index structure (a leaf search tree) that does not store keys in interior nodes. For brevity, from now on we use the term *search tree* to denote a tree with the properties just described. We enumerate the *levels* of nodes in a search tree from the root to the leaves: The root is at level 0, and the children of a node at level i are at level $i+1$. The *height h(T)* of a search tree T is the highest level number of a node in T. We denote with $\mathcal{K}_T(v) \subseteq \mathcal{K}$ the key interval represented by a node v in T, that is the set of all keys in $\mathcal{K}$ whose search path pass through v. The nature of $\mathcal{K}$ and the choice of the routers in the interior nodes imply that each such set $\mathcal{K}_T(v)$ is infinite.

A search tree is called *distributed*, if each node v is associated with a positive integer $j(v)$, the *server number* of v. For a distributed search tree T and a path $p = v_0, \ldots, v_s$ from the root to a leaf, the *distributed length $dl(p)$ of the path p* is the number of changes of the servers between adjacent nodes; more formally, $dl(v_0, \ldots, v_s) = |\{0 \leq i < s | j(v_i) \neq j(v_{i+1})\}|$. The *distributed height $dh(T)$* is the maximum of all distributed lengths of paths in T. Our interest in the distributed search tree height comes from the fact that in case the nodes on a path in T whose distributed length is $dh(T)$ are associated with $dh(T)$ different servers, answering a request may cost at least $dh(T)$ messages. That is, $dh(T)$ is then a lower bound on the number of messages for an operation in the worst case. The proof of Theorem 4.1 will show that in the worst case, in a distributed search tree that is generated by a stable method and stores m keys, there is a path whose distributed length is proportional to $\sqrt{m}$ and whose nodes are associated with just that many servers. Therefore, in such a distributed search tree for m keys, a request does need $\Omega(\sqrt{m})$ messages in the worst case.

Now, we formalize the idea that a sequence of insertions of keys according to a certain method creates a distributed search tree. Let $b \in \mathbf{N}$ be the block capacity, that is the maximal number of keys each server can hold (b is the same for all servers). A *distribution method M* is a process which maps each sequence $k_1, \ldots, k_m (m \in \mathbf{N})$ of pairwise different (p.d., for short) keys in $\mathcal{K}$ to a distributed search tree $T_M(k_1, \ldots, k_m)$ in such a way that for each server, the search path of at most b of the keys $k_1, \ldots, k_m$ ends at one of the leaves assigned to that server.

For a distribution method M and an integer $m \in \mathbf{N}$, let $dh_M(m)$ be the maximal distributed height of all search trees $T_M(k_1, \ldots, k_\mu)(k_1, \ldots, k_\mu \in \mathcal{K}$ p.d., $\mu \leq m)$ that M can generate by inserting at most m keys (let this value denote infinity, if no finite maximum exists).

Let us now formalize the search tree concept discussed previously. We call M *stable*, if for all $m \in \mathbf{N}$, for all p.d. keys $k_1, \ldots, k_m, k_{m+1} \in \mathcal{K}$, and for each node $v \in T_M(k_1, \ldots, k_m)$, there exist nodes $v_1, \ldots, v_n \in T_M(k_1, \ldots, k_m, k_{m+1})$ with $j(v_1) = \ldots = j(v_n) = j(v)$ and $\mathcal{K}_{T_M(k_1,\ldots,k_m)}(v) \subseteq \mathcal{K}_{T_M(k_1,\ldots,k_m,k_{m+1})}(v_1) \cup \ldots \cup \mathcal{K}_{T_M(k_1,\ldots,k_m,k_{m+1})}(v_n)$.

Our interest is in the degree of balancing that distributed search trees resulting from a stable method can achieve. For a real valued function $f : \mathbf{N} \to \mathbf{R}$, we call a distribution method M *f-dh-balanced*, if there is a constant $c_M > 0$ with $dh_M(m) \leq c_M f(m)$ for all $m \in \mathbf{N}$ (that is, $dh_M \in O(f)$).

4 A lower bound

Theorem 4.1 *Let $f : \mathbf{N} \to \mathbf{R}$ be a function with $f(m) = o(\sqrt{m})$. There exists no stable, f-dh-balanced distribution method.*

Proof:
Assume to the contrary that such a method M exists. The idea is to let an adversary insert keys whose search paths end in a leaf for which the distributed path length is longest.

To do so, let $k_1 \in \mathcal{K}$ be any key and $T^0 := T_M(k_1)$ the search tree that M generates. Let l^0 be the leaf in T^0 to which the search path for k_1 is guided; that is, $k_1 \in \mathcal{K}_{T^0}(l^0)$. For an example for T^0 and some further search trees in this proof, see Fig. 4.

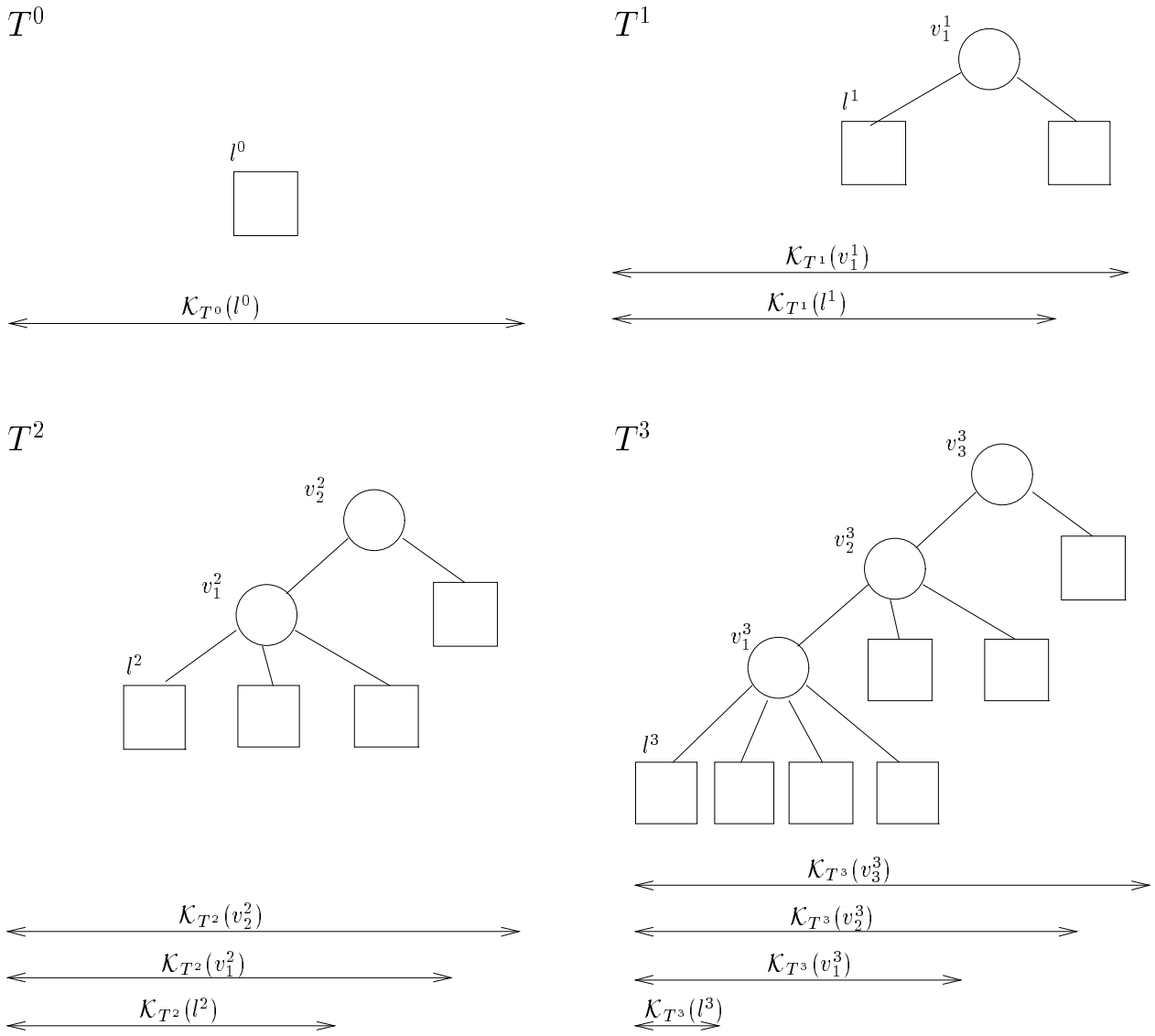

Figure 4: An example for a splitting history.

Now we choose b pairwise different keys $k_2, \ldots, k_{b+1} \in \mathcal{K}_{T^0}(l^0)$, all of them different from k_1 (this is possible, since $\mathcal{K}_{T^0}(l^0)$ is infinite), and consider the search tree $T^1 := T_M(k_1, k_2, \ldots, k_{b+1})$. T^1 has at least one leaf l^1 whose key set $\mathcal{K}_{T^1}(l^1)$ is not disjoint from $\mathcal{K}_{T^0}(l^0)$, and whose server number $j(l^1) \neq j(l^0)$. Since M is stable and $\mathcal{K}_{T^1}(l^1) \cap \mathcal{K}_{T^0}(l^0) \neq \emptyset$, there is a node v_1^1 in T^1 with $j(l^0) = j(v_1^1) \wedge \mathcal{K}_{T^1}(l^1) \cap \mathcal{K}_{T^1}(v_1^1) \neq \emptyset$. Because $j(l^1) \neq j(v_1^1)$, the node v_1^1 is different from l^1. Since l^1 and v_1^1 are nodes in the same search tree and l^1 is a leaf, v_1^1 must be a node on the path in T^1 from the root to l^1, that is, $\mathcal{K}_{T^1}(l^1) \subseteq \mathcal{K}_{T^1}(v_1^1)$. This implies that the distributed height

$dh(T_M(k_1, k_2, \ldots, k_{b+1}))$ is at least 1.

Now we continue by choosing $2b$ keys $k_{b+2}, \ldots, k_{3b+1} \in \mathcal{K}_{T^1}(l^1)$ in such a way that the keys $k_1, k_2, \ldots, k_{3b+1}$ are pairwise different. Since M is a distribution method, the search tree $T^2 := T_M(k_1, k_2, \ldots, k_{3b+1})$ has at least one leaf l^2, for which $\mathcal{K}_{T^2}(l^2) \cap \mathcal{K}_{T^1}(l^1) \neq \emptyset$, $j(l^2) \neq j(l^1)$ and $j(l^2) \neq j(v_1^1)$. Since M is stable, $\mathcal{K}_{T^2}(l^2) \cap \mathcal{K}_{T^1}(l^1) \neq \emptyset$ and $\mathcal{K}_{T^2}(l^2) \cap \mathcal{K}_{T^1}(v_1^1) \neq \emptyset$, there are nodes $v_1^2, v_2^2 \in T^2$ with $j(l^1) = j(v_1^2) \wedge \mathcal{K}_{T^2}(l^2) \cap \mathcal{K}_{T^2}(v_1^2) \neq \emptyset$ and $j(v_1^1) = j(v_2^2) \wedge \mathcal{K}_{T^2}(l^2) \cap \mathcal{K}_{T^2}(v_2^2) \neq \emptyset$. Since $j(l^2), j(v_1^2)$ and $j(v_2^2)$ are pairwise different, v_1^2 and v_2^2 are different interior nodes on the path from the root to leaf l^2 in T^2, that is, either $\mathcal{K}_{T^2}(l^2) \subseteq \mathcal{K}_{T^2}(v_1^2) \subseteq \mathcal{K}_{T^2}(v_2^2)$ or $\mathcal{K}_{T^2}(l^2) \subseteq \mathcal{K}_{T^2}(v_2^2) \subseteq \mathcal{K}_{T^2}(v_1^2)$ holds. It follows that the distributed height $dh(T_M(k_1, k_2, \ldots, k_{3b+1}))$ is at least 2.

Now we continue by choosing $3b$ keys $k_{3b+2}, \ldots, k_{6b+1} \in \mathcal{K}_{T^2}(l^2)$ in such a way that $k_1, \ldots, k_{6b+1}$ are pairwise different; by arguments extended from those given above, we get a search tree $T^3 := T_M(k_1, k_2, \ldots, k_{6b+1})$ with a distributed height of at least 3.

In general, in the sth step we choose sb keys in $\mathcal{K}_{T^{s-1}}(l^{s-1})$, and we get a search tree $T^s := T_M(k_1, k_2, \ldots, k_{m(s)})$ with a distributed height of at least s, where

$$m(s) = 1 + \sum_{i=1}^{s} ib = 1 + \frac{s(s+1)}{2}b.$$

That leads to

$$dh_M\Big(1 + \frac{s(s+1)}{2}b\Big) \geq s \text{ for all } s \in \mathbf{N}.$$

Since dh_M is monotonously increasing, and $1 + \frac{s(s+1)}{2}b \leq s^2 b$ holds for $s \geq 2$, we get

$$dh_M(s^2 b) \geq s \text{ for all } s \geq 2,$$

that is,

$$dh_M(m) \geq \sqrt{\frac{m}{b}} \text{ for an infinite number of values } m \in \mathbf{N}.$$

This contradicts the assumption that M is f-dh-balanced. □

5 A method that reaches the lower bound

The purpose of the next theorem is to show that the given bound of $\sqrt{m}$ in Theorem 4.1 is tight.

Theorem 5.1 *There exists a stable, $\sqrt{m}$-dh-balanced distribution method.*

Proof:
We prove this theorem by proposing a stable, $\sqrt{m}$-dh-balanced distribution method M. Let us first describe the method.

For $k_1 \in \mathcal{K}$, let $T_M(k_1)$ consist of only one leaf with server number 1.

Now, we inductively define the search trees that method M generates. Let $k_1, \ldots, k_m, k_{m+1} \in \mathcal{K}$ be pairwise different. $T_M(k_1, \ldots, k_m, k_{m+1})$ is generated from $T_M(k_1, \ldots, k_m)$ in the following way. Let $v_0, \ldots, v_s = l$ be the nodes on the search path for the key k_{m+1} in $T_M(k_1, \ldots, k_m)$. We denote the interval boundaries and the routers stored in v_{s-1}, the father of l (if it exists), by $\bar{k}_0, \ldots, \bar{k}_{\bar{r}}$, and we assume that the ith pointer of v_{s-1} is pointing to l. Now, we distinguish the following cases.

Case 1: [For less than b of the keys $k_1, \ldots, k_m$, the search path in $T_M(k_1, \ldots, k_m)$ ends in l.]
We define $T_M(k_1, \ldots, k_m, k_{m+1}) := T_M(k_1, \ldots, k_m)$.

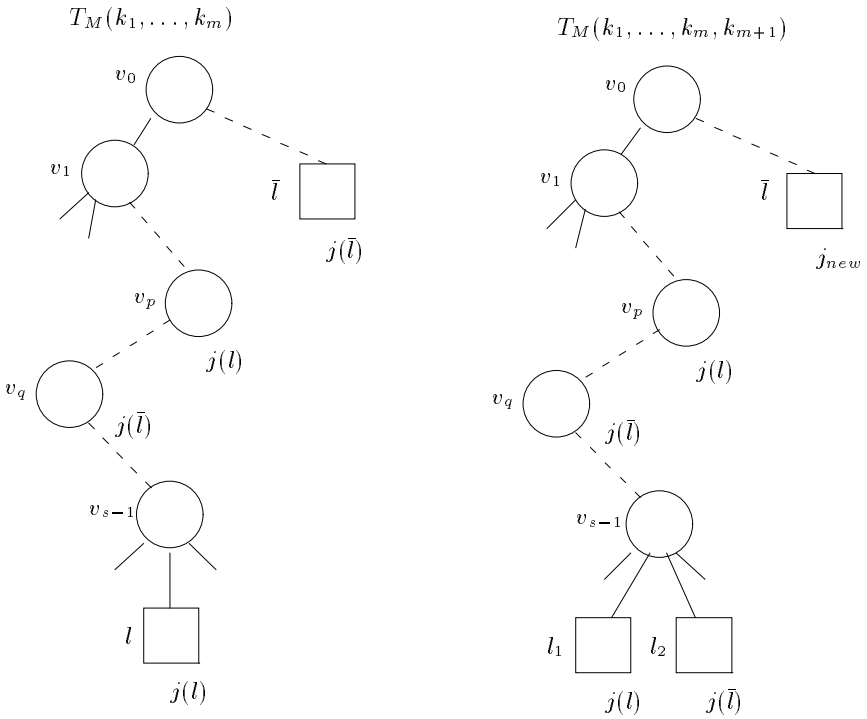

Figure 5: An illustration of the method M, Case 2.1.1.

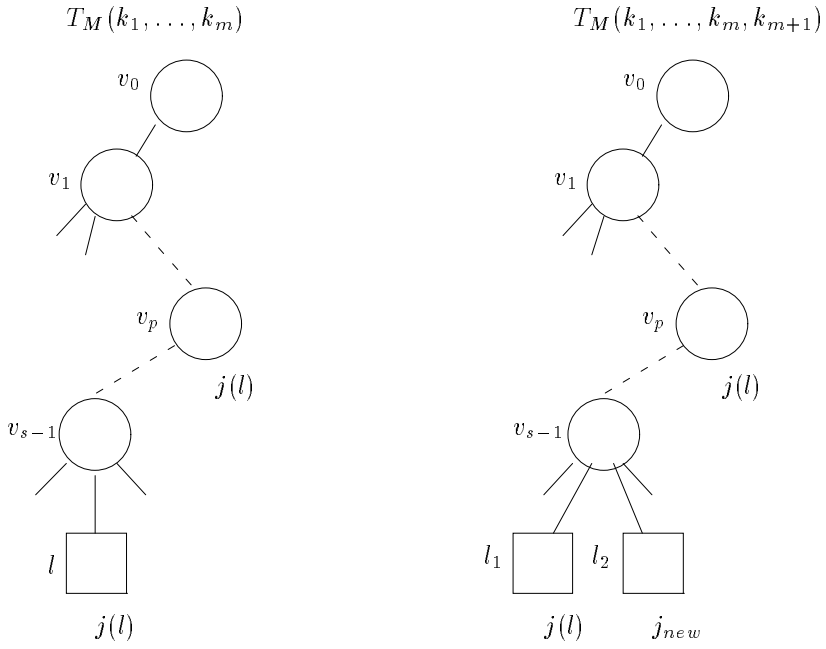

Figure 6: An illustration of the method M, Case 2.1.2.

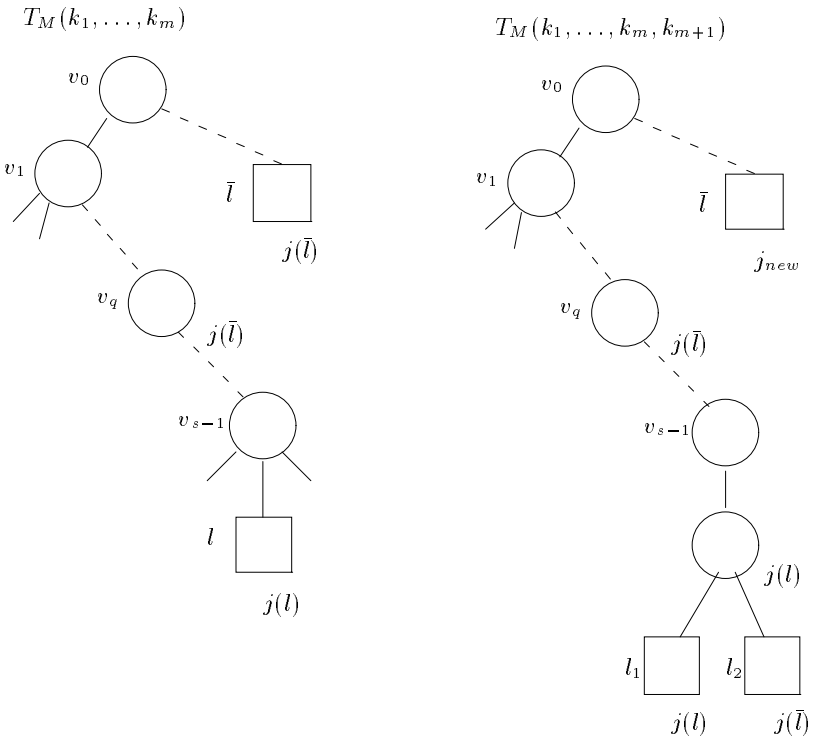

Figure 7: An illustration of the method M, Case 2.2.1.

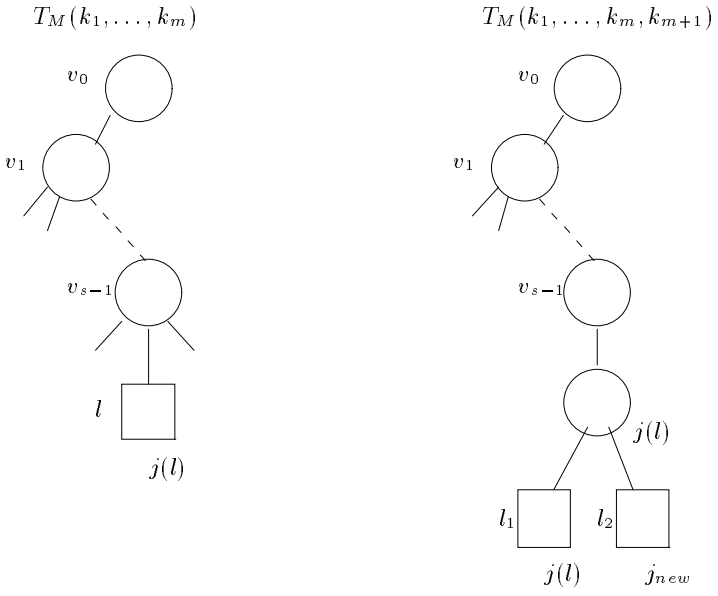

Figure 8: An illustration of the method M, Case 2.2.2.

Case 2: [For b of the keys $k_1, \ldots, k_m$, the search path in $T_M(k_1, \ldots, k_m)$ ends in l.[1]]

Case 2.1: [$s \geq 1$ and $j(l) = j(v_p)$ holds for some $0 \leq p \leq s-1$.]
We define $T_M(k_1, \ldots, k_m, k_{m+1})$ by changing $T_M(k_1, \ldots, k_m)$ in the following way. We split l into two leaves l_1 and l_2, with father v_{s-1}; that is, the ith pointer of v_{s-1} is pointing to l_1, and an additional pointer between the ith and the $(i+1)$th pointer is added, pointing to l_2. Further, an additional router k between $\bar{k}_{i-1}$ and $\bar{k}_i$ is added. We choose k as a median of the keys $\mathcal{K}(l) \cap \{k_1, \ldots, k_{m+1}\}$[2]. In addition, we change some server numbers of $T_M(k_1, \ldots, k_m)$. For doing so, we distinguish two cases.

Case 2.1.1: [There is a leaf $\bar{l}$ in $T_M(k_1, \ldots, k_m)$ with the properties that

- the level of $\bar{l}$ is smaller than the level of l,
- there is a node $v_q \in \{v_0, \ldots, v_{s-1}\}$ with $j(v_q) = j(\bar{l})$.]

We set $(j(\bar{l}), j(l_1), j(l_2)) := (j_{new}, j(l), j(\bar{l}))$, where j_{new} is a server number that does not occur in the search tree $T_M(k_1, \ldots, k_m)$.

Case 2.1.2: [There is no such leaf $\bar{l}$ with the properties required in case 2.1.1]
We set $(j(l_1), j(l_2)) := (j(l), j_{new})$.

Case 2.2: [$s = 0$ or ($s \geq 1$ and $j(l) \neq j(v_p)$ holds for all $0 \leq p \leq s-1$)]
Here, l becomes an interior node storing the interval boundaries $\bar{\bar{k}}_0$ and $\bar{\bar{k}}_2$ and the router $\bar{\bar{k}}_1$ with $\bar{\bar{k}}_0 = -\infty, \bar{\bar{k}}_1 = k$ and $\bar{\bar{k}}_2 = \infty$, if $s = 0$, resp. $\bar{\bar{k}}_0 = \bar{k}_{i-1}, \bar{\bar{k}}_1 = k$ and $\bar{\bar{k}}_2 = \bar{k}_i$, if $s \geq 1$, where k is chosen as in Case 2.1. l has two pointers to two new leaves l_1 and l_2. As in Case 2.1, we distinguish two cases for changing some server numbers of $T_M(k_1, \ldots, k_m)$.

Case 2.2.1: [$s \geq 1$ and there is a leaf $\bar{l}$ in $T_M(k_1, \ldots, k_m)$ with the properties that

- the level of $\bar{l}$ is smaller than or equal to the level of l,
- there is a node $v_q \in \{v_0, \ldots, v_{s-1}\}$ with $j(v_q) = j(\bar{l})$.]

We set $(j(\bar{l}), j(l_1), j(l_2)) := (j_{new}, j(l), j(\bar{l}))$.

Case 2.2.2: [$s = 0$ or ($s \geq 1$ and there is no such leaf $\bar{l}$ with the properties required in case 2.2.1)]
We set $(j(l_1), j(l_2)) := (j(l), j_{new})$.

This completes the description of the method M. To prove that M is a distribution method, stable and $\sqrt{m}$-dh-balanced, we will prove several properties for all search trees $T_M(k_1, \ldots, k_m)$ ($k_1, \ldots, k_m \in \mathcal{K}$ p.d., $m \in \mathbf{N}$) by induction.

Lemma 5.1 *For any leaf l of a search tree $T_M(k_1, \ldots, k_m)$ ($k_1, \ldots k_m \in \mathcal{K}$ p.d., $m \in \mathbf{N}$) generated by method M, for which an interior node $v \in T_M(k_1, \ldots, k_m)$ exists with $j(v) = j(l)$, there exists a node $\bar{v} \in T_M(k_1, \ldots, k_m)$ on the path from the root of $T_M(k_1, \ldots, k_m)$ to l for which $j(\bar{v}) = j(l)$.*

Proof:
By induction. It is clear that the statement is true for any search tree $T_M(k_1)$ with $k_1 \in \mathcal{K}$. Now, we assume that the statement is true for all search trees $T_M(k_1, \ldots, k_m)$ with $k_1, \ldots, k_m \in$

[1] It is not possible that for more than b of the keys $k_1, \ldots, k_m$, the search path in $T_M(k_1, \ldots, k_m)$ ends in l, since the method M will turn out to be a distribution method.

[2] A median of r pairwise different keys $k_1, \ldots, k_r$ (w. l. o. g. given as an ordered sequence) is the key $k_{\frac{r+1}{2}}$, if r is odd, and a key bigger than $k_{\frac{r}{2}}$ and smaller than $k_{\frac{r}{2}+1}$, if r is even.

$\mathcal{K}$ p.d., where m is fixed. We choose p.d. $k_1, \ldots, k_m, k_{m+1}$ in $\mathcal{K}$. If $T_M(k_1, \ldots, k_m, k_{m+1})$ is generated from $T_M(k_1, \ldots, k_m)$ by Case 1, Case 2.1.2 or Case 2.2.2, the argument is obvious. If it is generated from $T_M(k_1, \ldots, k_m)$ according to Case 2.1.1 or Case 2.2.1, the only leaves in $T_M(k_1, \ldots, k_m, k_{m+1})$ for which we have to check the condition are $\bar{l}, l_1$ and l_2. Since the server number of $\bar{l}$ in $T_M(k_1, \ldots, k_m, k_{m+1})$ is j_{new}, there is nothing to prove. For l_1 the condition holds, because it holds for l in $T_M(k_1, \ldots, k_m)$. The condition holds for l_2, because v_q lies on the path from v_1 to l_2 in $T_M(k_1, \ldots, k_m, k_{m+1})$ and has the same server number as l_2. □

Lemma 5.2 *The server numbers of the interior nodes of any path in any search tree* $T_M(k_1, \ldots, k_m)(k_1, \ldots, k_m \in \mathcal{K}$ *p.d.*, $m \in \mathbf{N})$ *are pairwise different.*

Proof:
By induction. The statement is true for $m = 1$ and all $k_1 \in \mathcal{K}$. Now, we assume that it is true for a fixed m and all p.d. $k_1, \ldots, k_m$. We choose $k_{m+1} \in \mathcal{K}$ p.d. from $k_1, \ldots, k_m$ and consider $T_M(k_1, \ldots, k_m, k_{m+1})$. If $T_M(k_1, \ldots, k_m, k_{m+1})$ is generated from $T_M(k_1, \ldots, k_m)$ according to Case 1 or Case 2.1, there is nothing to prove. If it is generated from $T_M(k_1, \ldots, k_m)$ according to Case 2.2, the condition of the case makes sure that for the new interior node v in $T_M(k_1, \ldots, k_m, k_{m+1})$ created from l in $T_M(k_1, \ldots, k_m)$, there is no interior node on the path in $T_M(k_1, \ldots, k_m, k_{m+1})$ from the root to v with the same server number as v. □

Lemma 5.3 *Let* l *be any leaf of a search tree* $T_M(k_1, \ldots, k_m)(k_1, \ldots, k_m \in \mathcal{K}$ *p.d.*, $m \in \mathbf{N})$ *on the level with the highest level number, where* $j(l) \neq j(v_i)$ *for all* $0 \leq i < s$, *with* $v_0, \ldots, v_{s-1}$ *denoting the nodes on the path from the root to* l *in* $T_M(k_1, \ldots, k_m)$, *if* $s \geq 1$. *Then there are at least* s *further leaves on the level with the highest level number in* $T_M(k_1, \ldots, k_m)$.

Proof:
By induction. For $m = 1$, there is nothing to prove. Let $k_1, \ldots, k_m \in \mathcal{K}$ p.d., $m \in \mathbf{N}$, choose $k_{m+1} \in \mathcal{K}$ p.d. from $k_1, \ldots, k_m$, and consider $T_M(k_1, \ldots, k_m, k_{m+1})$. If $T_M(k_1, \ldots, k_m, k_{m+1})$ is generated from $T_M(k_1, \ldots, k_m)$ according to Case 1, there is nothing to prove. If it is generated according to Case 2.1.1 or 2.2.1, the only leaves in $T_M(k_1, \ldots, k_m, k_{m+1})$ we must consider are $\bar{l}, l_1, l_2$. But none of these leaves satisfies the properties of l in lemma 5.3, because $\bar{l}$ is not on the level with highest level number of $T_M(k_1, \ldots, k_m, k_{m+1})$, and for l_1 and l_2 there are nodes on the paths from the root to these nodes in $T_M(k_1, \ldots, k_m, k_{m+1})$ with the same server numbers. If Case 2.1.2 or Case 2.2.2 applies, we have to check the statement only for $l_2 \in T_M(k_1, \ldots, k_m, k_{m+1})$ and only in the case that l_2 is lying on the highest level of $T_M(k_1, \ldots, k_m, k_{m+1})$. Denote by $v_0, \ldots, v_{s-1} \in T_M(k_1, \ldots, k_m)$ the nodes on the path from the root to leaf l in $T_M(k_1, \ldots, k_m)$. For $s = 0$, the proof is immediate. If $s \geq 1$, Lemma 5.2 implies that the server numbers of $v_0, \ldots, v_{s-1}$ are pairwise different. Since it is easy to see that for each interior node a leaf with the same server number exists, there must be at least s leaves on a level higher than or equal to that of $l_2 \in T_M(k_1, \ldots, k_m, k_{m+1})$, just because Case 2.1.2 resp. 2.2.2 applies. All these leaves must lie on the same level as l_2, because l_2 lies on the level with highest level number. This completes the proof. □

Lemma 5.4 *Each search tree* $T_M(k_1, \ldots, k_m)$ $(k_1, \ldots, k_m \in \mathcal{K}$ *p.d.*, $m \in \mathbf{N})$ *has at least* $i+1$ *nodes at the* i*th level, for all* $0 \leq i < h(T_M(k_1, \ldots, k_m))$.

Proof:
By induction. For $m = 1$, the proof is obvious. Now assume that the statement is true for any p.d. $k_1, \ldots, k_m \in \mathcal{K}$, and choose $k_{m+1} \in \mathcal{K}$ p.d. from $k_1, \ldots, k_m$. If $T_M(k_1, \ldots, k_m, k_{m+1})$ is generated

from $T_M(k_1,\ldots,k_m)$ according to Case 1 or Case 2.1, there is nothing to prove. If Case 2.2 is applied and $l \in T_M(k_1,\ldots,k_m)$ does not lie on the level with highest level number, the proof is also immediate. If Case 2.2 is applied and $l \in T_M(k_1,\ldots,k_m)$ lies on the level with highest level number, Lemma 5.3 implies that there are at least s further leaves on the highest level s of $T_M(k_1,\ldots,k_m)$, and therefore level s of $T_M(k_1,\ldots,k_m,k_{m+1})$ contains at least $s+1$ nodes. For the lower levels of $T_M(k_1,\ldots,k_m,k_{m+1})$, the condition holds because it holds for $T_M(k_1,\ldots,k_m)$. □

Let us now conclude the proof Theorem 5.1 by proving the three properties of M.

M is a distribution method: It follows immediately by induction that each server is assigned to at most one leaf. Thus, because within a split, the split key is chosen accordingly, it follows by induction that M is a distribution method.

M is stable: We show that for all keys $k_1,\ldots,k_m,k_{m+1} \in \mathcal{K}$ p.d., $m \in \mathbf{N}$, and for each node $v \in T_M(k_1,\ldots,k_m)$ a node $\bar{v} \in T_M(k_1,\ldots,k_m,k_{m+1})$ exists with $j(v) = j(\bar{v})$ and $\mathcal{K}_{T_M(k_1,\ldots,k_m)}(v) \subseteq \mathcal{K}_{T_M(k_1,\ldots,k_m,k_{m+1})}(\bar{v})$. This is obvious, if $T_M(k_1,\ldots,k_m,k_{m+1})$ is generated from $T_M(k_1,\ldots,k_m)$ according to Case 1, Case 2.1.2 or Case 2.2.2. If Case 2.1.1 or Case 2.2.1 is applied, the statement is true for l in $T_M(k_1,\ldots,k_m)$, since $\mathcal{K}_{T_M(k_1,\ldots,k_m)}(l) \subseteq \mathcal{K}_{T_M(k_1,\ldots,k_m,k_{m+1})}(v_q)$, and it follows from Lemma 5.1 for $\bar{l} \in T_M(k_1,\ldots,k_m)$. For all other nodes in $T_M(k_1,\ldots,k_m)$, there is nothing to prove.

M is $\sqrt{m}$-dh-balanced: Consider a tree $T_M(k_1,\ldots,k_m)$ $(k_1,\ldots,k_m \in \mathcal{K}$ p.d., $m \in \mathbf{N})$ with height $h := h(T_M(k_1,\ldots,k_m))$. Lemma 5.4 implies that the

$$\begin{aligned} \text{number of nodes of } T_M(k_1,\ldots,k_m) &\geq \sum_{i=0}^{h-1}(i+1) \\ &= \frac{h(h+1)}{2} \\ &\geq \frac{h^2}{2}. \end{aligned}$$

Since at least half of the nodes of $T_M(k_1,\ldots,k_m)$ are leaves (proved by an easy inductive argument), we get

$$\text{number of leaves of } T_M(k_1,\ldots,k_m) \;\geq \frac{h(T_M(k_1,\ldots,k_m))^2}{4}.$$

That is,

$$h(T_M(k_1,\ldots,k_m)) \leq 2 \cdot \sqrt{\text{number of leaves of } T_M(k_1,\ldots,k_m)}.$$

Since the method M guarantees that for each leaf in $T_M(k_1,\ldots,k_m)(m \geq \frac{b}{2})$, the search paths of at least $\frac{b}{2}$ of the keys $k_1,\ldots,k_m$ end in that leaf, we get

$$\text{number of leaves of } T_M(k_1,\ldots,k_m) \;\leq \frac{m}{\frac{b}{2}} \text{ for all } m \geq \frac{b}{2},$$

and therefore,

$$h(T_M(k_1,\ldots,k_m)) \leq 2\sqrt{\frac{2m}{b}} = c_M\sqrt{m} \text{ for all } m \geq \frac{b}{2},$$

where $c_M = 2\sqrt{\frac{2}{b}}$ is constant. Since the distributed height is smaller than or equal to the tree height, we obtain

$$dh(T_M(k_1,\ldots,k_m)) \leq c_M\sqrt{m}$$

for all trees $T_M(k_1,\ldots,k_m)(m \geq \frac{b}{2})$, and thus

$$dh_M(m) \leq c_M\sqrt{m}$$

for all $m \geq \frac{b}{2}$. Since $dh_M(m) = 1$ for $m < \frac{b}{2}$, the proof is complete (by choosing $c_M := \max(c_M, 1)$). □

6 Discussion

We have shown that the distributed height of a distributed search tree resulting from a stable distribution method is $\Omega(\sqrt{m})$ in the worst case for m keys. In addition, we have shown that there is a stable distribution method satisfying the bound of $O(\sqrt{m})$ on the distributed height of its distributed search trees. Hence, these bounds are tight.

The method M proposed in the proof of Theorem 5.1 does not fully describe a distributed dictionary: It only describes the development of the virtual, global tree over time by describing how the tree grows and how servers are associated with nodes. This does not define which site knows which part of the virtual, global tree, which messages must be forwarded to which servers, and how we can make sure that eventually, each request arrives at the correct server. It also does not define a mechanism of correction messages that bring the obsolete knowledge of a site up to date. The full description of a suitable method is not within the focus of this paper, since we are interested here mainly in a step towards a theoretical foundation of distributed data structures. It may suffice to state that a method using no more than $O(\sqrt{m})$ messages per operation can actually be derived ([4]). On the constructive side, our result implies that we will have to drop the stability requirement in order to arrive at fast distributed search trees. Litwin et al. [6] do so for their distributed variant of a B-tree, and they indeed achieve logarithmic height. A search operation, however, does not simply follow a path from the root to a leaf; it may need to go back up in the tree to the parent of a visited node. As a consequence, Litwin et al. [6] do not give a bound on the number of messages. In addition, the loss of the straight guiding property may impede scalability: Sending messages to father nodes may distribute the message handling load over the servers more unevenly, with a disadvantage for servers whose nodes are closer to the global root.

Acknowledgement

We wish to thank Thomas Ottmann for many fruitful and inspiring discussions.

References

[1] R. Devine: *Design and Implementation of DDM: A Distributed Dynamic Hashing Algorithm.* 4th Int. Conference on Foundations of Data Organization on Algorithms FODO, 1993.

[2] B. Kröll, P. Widmayer: *Distributing a Search Tree among a Growing Number of Processors.* Proc. ACM SIGMOD Conference on the Management of Data, 1994, 265 – 276.

[3] B. Kröll, P. Widmayer: *Balanced Distributed Search Trees Do Not Exist.* Proceedings of the WADS Conference, 1995.

[4] B. Kröll, P. Widmayer: manuscript, in preparation, 1995.

[5] W. Litwin, M.–A. Neimat, D.A. Schneider: *LH* — Linear Hashing for Distributed Files.* Proc. ACM SIGMOD Conference on the Management of Data, 1993, 327 – 336.

[6] W. Litwin, M.–A. Neimat, D.A. Schneider: *RP* — A Family of Order Preserving Scalable Distributed Data Structures.* Proc. of VLDB, 1994, 342 – 353.

[7] R. Vingralek, Y. Breitbart, G. Weikum: *Distributed File Organization with Scalable Cost/Performance.* Proc. ACM SIGMOD Conference on the Management of Data, 1994, 253 – 264.